Mobil- és webes szoftverek - 3. előadás

Activity váltás

- A indít B-t
- A → onPause() → itt mentsen olyat adatbázisba, amit B olvashat
- B → onCreate(), onStart() és onResume()
- A → onStop()

Activity Back Stack

- Back Stack tetején: ami épp a kijelzőn van, a legutóbbik alatta sorban
- Splash-en hívunk finish-t és akkor az nem kerül back stack-re
    - vagy manifest-ben beállítunk egy megfelelő propretí-t

Új Activity indítása

```
fun runSecondActivity() {
    val myIntent: Intent = Intent()
    myIntent.setClass(this@MainActivity, SecondActivity::class.java) //manifest-
ből ismerit az osztály
    //Adat átadása
    myIntent.putExtra("KEY_DATA", "Hi there!")
    startActivity(myIntent)
}
```

Felhasználói felület ablakok

- **Képernyő méret** (*screen size*) → fizikai képátló
    - 4 kategória: small, normal, large, extra large
- **Képernyő sűrűség** (*screen density*) → pixelek száma egy adott fizikai területen belül
    - általában inch-enkénti képpont (*dpi - dots per inch*)
    - ennek is van mindenféle kategóriája (low, medium, ...)
- **Orientáció** (*orientation*)
    - álló (*portrait*) és fekvő (*landscape*)
- **Felbontás** (*resolution*) → pixelek száma a képernyőn (*px*)
- **Sűrűség független pixel** (density-independent pixel) → (*dp*)
    - UI tervezéskor EZT használjuk!
    - 1 dp = fizikai pixel egy 160 dpi-s képernyőn
    - px = dp * (dpi / 160)
- **Sűrűség független méretezés szövegekhez** (*sp*)
    - plusz: ha átállítjuk beállításba, mert vaksik vagyunk, akkor ez igazodik ahoz

Erőforrás típusok

- res mappákban

    - hangok, videók, képek, dinamikus XML drawable, animációk, stílusok, témák, szöveges erőforrások, stb
- szöveges erőforrások → ras/values/strings.xml
    - többnyelvűség így biztosítható
    - paraméterezhetőség

```
<string name="timeFormat">%1$d minutes ago</string> //ez van xml-ben
Context.getString(R.strings.timeFormat, 14) //ez van a forrásfájlban
```

    - futási időben választ megfelelő erőforrást az alkalmazás (-hu, -large, stb)
- **Sűrűség függetlenség** (pixelsűrűségre vonatkozik)
    - azaz a felhasználói eleme a felhasználó szemszögéből megőrzik a fizikai méretüket különböző sűrűségeken
    - Android két módon segíti ezt
        - dp (sűrűségfüggetlen pixel) kiszámítása alapján skálázza a UI-t
        - automatikusan átskálázza a kép erőforrásokat
    - amiket ajánlott használni:
        - `wrap_content`, `match_parent`, `dp` értékek
        - súlyozás
        - pixelben megadás kerülése!
        - AbsoluteLayout elkerülése!
        - különböző képerőforrások különböző képernyősűrűségekhez
        - szövegek méretezése sp-vel

Felhasználói felület felépítése

- Minden ami megjelenik → View-ból származnak le
    - Layout-ok is a View-ból (View Group-ból) származnak le → FindViewById() függvény van a háttérbe
    - ViewGroup-ok egymásba ágyazhatók

Layout-ok

- **LinearLayout**
    - `gravity` → merre legyenek rendezve
    - `visibility` →
        - *visible* - látszik
        - *invisible* - nem látszik, de helyet foglal
        - *gone* - helyet se foglal
    - `weightSum`-mal teljes súly megadható (ennyi részre osztja)
        - `layout_height="0dp"`-t kell ilyenkor megadni
        - `layout_weight` → a teljes súly annyiad részét fogja kitenni (túl nagy érték adásakor kicsúszunk a kijelző méretéből)
        - belső layout-nak is lehet saját összsúlya!
    - külön layout fájlból is kiszervezhető egy rész
        - `<include layout="@layout/layout_internal" />`
- **ScrollView** → ami benne van, görgethetős lesz

- **RelativeLayout**
  - már ritkábban használjuk (Constraint-et használunk helyette)
  - elemek egymáshoz vagy az ős layout-hoz képesti pozícióját lehet beállítani
- **ConstraintLayout** (~iOS AutoLayout) → ez a legjobb
  - flat view hierarchia (ne legyenek egymásba ágyazott layout-ok) → gyorsabb program
  - igazodási szabályokat adunk meg
    - horizontális és vertikális constraint
    - szabály = kapcsolat → másik view-hez képet, szülőhöz képest, láthatatlan sorvezetőhöz képest (guideline)
    - ellentétes szabályok → legjobb verzió (jobbra és balra is = középre)
    - `layout_width = "0dp"` = match constraint → teljes kitöltés
  - guideline → ehhez is igazodhatnak
    - ráhelyezés = bal és jobb oldala is igazodjon hozzá
- **AbsoluteLayout** (ezt NE használjuk)
- **GridLayout** → ez is mellékes

Dinamikus UI kezelés - LayoutInflater

- Layout XML-ben készített UI példányosítása
  - ```
    val myView = getLayoutInflater().inflate(R.layout.data_row, null, false)
    //létrehoz egy data_row példányát
    viewRodo.tvData.text = etTodo.text.toString() //beállítjuk a szöveget
    layoutMain.addView(myView) //hozzáadjuk
    ```
- **Validáció** → setError(errorText)
- Menük
  - menü erőforrás kell

    ```
    <menu ...
        <item ... />
        ...
    </menu>
    ```
  - menü kezelése

    ```
    override fun onCreateOptionsMenu(menu: Menu?): Boolean { //menü
    létrehozása
        val inflater = menuInflater
        inflater.inflate(R.menu.mymenu, menu)
        return true
    }
    override fun onOptionsItemSelected(item: MenuItem): Boolean {
    //menüpontok
        if(item.itemId == R.id.action_start) {
            Toast.makeText(
            this,
            item.getTitle(), Toast.LENGTH_LONG
            ).show()
        }
        if(item.itemId == R.id.action_stop) {
    ```
    ```

```
            Toast.makeText(
            this,
            item.getTitle(), Toast.LENGTH_LONG
            ).show()
        }
        return super.onOptionsItemSelected(item)
    }
```